Programação Visual
Trabalho de Laboratório nº 9

Objetivo	MVC – Testes unitários usando a *framework* xUnit.
Programa	Definição e utilização de testes unitários na aplicação ESTeSoccer.
Regras	Implementar o código necessário e testar no fim de cada nível. Use as convenções de codificação adotadas para a linguagem C# e para o modelo MVC.

Nível 1

- Descarregue e descompacte o ficheiro "**Lab 9 – Materiais**" fornecido com este enunciado. Depois de o descompactar, irá encontrar a solução **ESTeSoccer**. Abra a solução, crie a migração e a base de dados associada e verifique que está a funcionar corretamente.
- Pretende-se agora testar a aplicação criada através de testes unitários. Sendo assim, adicione um novo projeto de testes **xUnit Test Project C#** com o nome **ESTeSoccerTest** e torne-o o **Startup Project**. Finalmente, adicione ao projeto a referência para o projeto **ESTeSoccer**.
- Altere a classe aberta **UnitTest1** para **HomeControllerTest** e defina nesta classe testes unitários para as ações **Index** e **Privacy** do controlador **Home**.
  Estes testes unitários irão servir para verificar se as páginas são abertas corretamente.
- Corra os testes criados.

Nível 2

- Pretende-se agora criar alguns testes unitários para o controlador **LeaguesController**. Sendo assim, comece por criar a classe de testes deste controlador e um teste unitário para a ação **Index**. Para ter um contexto de teste e poder instanciar o controlador, utilize uma base de dados **Sqlite** em memória.
  Nota: Para utilizar a base de dados **Sqlite** deve instalar o seguinte pacote:
  `Install-package microsoft.entityframeworkcore.sqlite`

- Para utilizar a base de dados em memória, terá de criar uma nova classe **ApplicationDbContextFixture** que extende a interface **IDisposable** e, após a criação deste, inclua 1 nova liga e implemente o método de limpeza **Dispose**.
  Nota: Atenção que já existe um método no contexto que já cria dados de teste na construção do contexto.
- Implemente a interface `IClassFixture<` **ApplicationDbContextFixture** `>` na classe **LeaguesController** e defina agora os testes para a ação **Details**. Sendo estes testes para várias possibilidades, com ligas inexistentes, com valores *null* e com uma liga existente.

Nível 3

- Pretende-se agora criar alguns testes unitários para o controlador **TeamsController**. Neste caso, reutilize o contexto **Sqlite** anteriormente criando, e adicione mais 2 equipas. Teste a ação **Index.**
- Crie ainda os testes das duas ações **Create** existentes no controlador.

Programação Visual
Trabalho de Laboratório nº 9

Nível 4

- Crie a classe de teste e os testes para a ação **Index** para o controlador **PlayersController**.
- Crie os testes da ação **PlayerExists** do mesmo controlador.
  Nota: Altere o método **PlayerExists** para que este seja acessível fora do controlador e utilize o atributo **NonAction** no método garantindo que este método não tenha o mesmo comportamento que as outras ações do controlador.

Nível 5

- Crie os testes para as ações **Delete** e **DeleteConfirmed** do controlador **PlayersController**.
- Crie os testes das ações **Edit**.

Desafio

- Crie agora uma classe nova de testes e efetue os testes unitários para o projeto **ESTeSoccerClient.**
- Devido às dependências da API ao projeto base, não será possível utilizar uma base de dados em memória, portanto garanta que no final de cada teste reverta as alterações à Base de Dados.
  Nota: Visto que irá utilizar a base de dados atual, irá ser necessário alterar a configuração do Startup Project de **SINGLE** para **MULTIPLE** e correr os projetos ao mesmo tempo sem *debugging* para conseguir completar os testes unitários.

Notas

Para os identificadores siga as convenções adotadas pelo C#, nomeadamente:

- A notação camelCase para o nome das variáveis locais e identificadores privados.
- A notação PascalCase para os nomes públicos dos métodos, classes e interfaces.
- Não utilize o símbolo '_' nos identificadores nem abreviaturas